IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant: Jeremy John CARROLL	)	Group: Not yet assigned
	)	Examiner: Not yet assigned
Serial No.: 10/644,273	)	Our Ref: B-5212 621174-8
Filed: August 19, 2003	)	
For: "PROCESSING OF DATA"	)	Date: November 5, 2003

OIPE JC112 NOV 10 2003 PATENT & TRADEMARK OFFICE

CLAIM TO PRIORITY UNDER 35 U.S.C. 119

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

[X] Applicants hereby make a right of priority claim under 35 U.S.C. 119 for the benefit of the filing date(s) of the following corresponding foreign application(s):

COUNTRY	FILING DATE	SERIAL NUMBER
GB	15 November 2002	0226697.1

[ ] A certified copy of each of the above-noted patent applications was filed with the Parent Application No.______________.

[X] To support applicant's claim, certified copies of the above-identified foreign patent applications are enclosed herewith.

[ ] The priority document will be forwarded to the Patent Office when required or prior to issuance.

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first-class mail in an envelope addressed to the "Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450", on November 5, 2003 by Ericca Long

[illegible]

Respectfully submitted,

[illegible]

Richard P. Berg
Attorney for Applicant
Reg. No. 28,145

LADAS & PARRY
5670 Wilshire Boulevard
Suite 2100
Los Angeles, CA 90036
Telephone: (323) 934-2300
Telefax: (323) 934-0202

10/644, 273

PATENTS • DESIGNS The Patent Office COPYRIGHT • TRADE MARKS

INVESTOR IN PEOPLE

The Patent Office
Concept House
Cardiff Road
Newport
South Wales
NP10 8QQ

I, the undersigned, being an officer duly authorised in accordance with Section 74(1) and (4) of the Deregulation & Contracting Out Act 1994, to sign and issue certificates on behalf of the Comptroller-General, hereby certify that annexed hereto is a true copy of the documents as originally filed in connection with the patent application identified therein.

In accordance with the Patents (Companies Re-registration) Rules 1982, if a company named in this certificate and any accompanying documents has re-registered under the Companies Act 1980 with the same name as that with which it was registered immediately before re-registration save for the substitution as, or inclusion as, the last part of the name of the words "public limited company" or their equivalents in Welsh, references to the name of the company in this certificate and any accompanying documents shall be treated as references to the name with which it is so re-registered.

In accordance with the rules, the words "public limited company" may be replaced by p.l.c., plc, P.L.C. or PLC.

Re-registration under the Companies Act does not constitute a new legal entity but merely subjects the company to certain additional company law rules.

Signed [illegible]

Dated 14 August 2003

15/11 '02 16:59 FAX → UK PO 001

10/644,273

Patents Form 1/77

Patents Act 1977
(Rule 16)

THE PATENT OFFICE
A
15 NOV 2002
RECEIVED BY FAX

The Patent Office (Patents · Designs · Copyright · Trade Marks)

THE PATENT OFFICE
A
15 NOV 2002
RECEIVED BY FAX

15NOV02 E763889-1 D01463
P01/7700 0.00-0226697.1

The Patent Office

Cardiff Road
Newport
South Wales
NP10 8QQ

Request for grant of a patent

(See the notes on the back of this form. You can also get an explanatory leaflet from the Patent Office to help you fill in this form)

1. Your reference	20030135-1 GB	
2. Patent application number *(The Patent Office will fill in this part)*	15 NOV 2002	0226697.1
3. Full name, address and postcode of the or of each applicant *(underline all surnames)*	Hewlett-Packard Company 3000 Hanover Street Palo Alto CA 94304, USA	
Patents ADP number *(if you know it)*		49658001
If the applicant is a corporate body, give the country/state of its incorporation	Delaware, USA	
4. Title of the invention	Processing of Data	
5. Name of your agent *(if you have one)*	ROBERT E SQUIERS ~~Bruce G R Jones~~	
"Address for service" in the United Kingdom to which all correspondence should be sent *(including the postcode)*	Hewlett-Packard Ltd, IP Section Filton Road, Stoke Gifford Bristol BS34 8QZ	79281870001
Patents ADP number *(if you know it)*		

	Country	Priority application number *(if you know it)*	Date of filing *(day / month / year)*
6. If you are declaring priority from one or more earlier patent applications, give the country and the date of filing of the or of each of these earlier applications and *(if you know it)* the or each application number			

	Number of earlier application	Date of filing *(day / month / year)*
7. If this application is divided or otherwise derived from an earlier UK application, give the number and the filing date of the earlier application		

8. Is a statement of inventorship and of right to grant of a patent required in support of this request? *(Answer 'Yes' if:* — Yes
a) any applicant named in part 3 is not an inventor, or
b) there is an inventor who is not named as an applicant, or
c) any named applicant is a corporate body.
See note (d))

Patents Form 1/77
0052827 15-Nov-02 04:05

Patents Form 1/77

9. Enter the number of sheets for any of the following items you are filing with this form. Do not count copies of the same document

Continuation sheets of this form	
Description	20
Claim(s)	2
Abstract	
Drawing(s)	1 + 1 [illegible]

10. If you are also filing any of the following, state how many against each item.

Priority documents	
Translations of priority documents	
Statement of inventorship and right to grant of a patent *(Patents Form 7/77)*	
Request for preliminary examination and search *(Patents Form 9/77)*	
Request for substantive examination *(Patents Form 10/77)*	
Any other documents *(please specify)*	

11. I/We request the grant of a patent on the basis of this application.

Signature [illegible] Date 15/11/2002

12. Name and daytime telephone number of person to contact in the United Kingdom

Warning

After an application for a patent has been filed, the Comptroller of the Patent Office will consider whether publication or communication of the invention should be prohibited or restricted under Section 22 of the Patents Act 1977. You will be informed if it is necessary to prohibit or restrict your invention in this way. Furthermore, if you live in the United Kingdom, Section 23 of the Patents Act 1977 stops you from applying for a patent abroad without first getting written permission from the Patent Office unless an application has been filed at least 6 weeks beforehand in the United Kingdom for a patent for the same invention and either no direction prohibiting publication or communication has been given, or any such direction has been revoked.

Notes

a) *If you need help to fill in this form or you have any questions, please contact the Patent Office on 08459 500505.*

b) *Write your answers in capital letters using black ink or you may type them.*

c) *If there is not enough space for all the relevant details on any part of this form, please continue on a separate sheet of paper and write "see continuation sheet" in the relevant part(s). Any continuation sheet should be attached to this form.*

d) *If you have answered 'Yes' Patents Form 7/77 will need to be filed.*

e) *Once you have filled in the form you must remember to sign and date it.*

f) *For details of the fee and ways to pay please contact the Patent Office.*

DUPLICATE

PROCESSING OF DATA

The present invention relates to the processing of data, such as for example data which is represented in graphical form.

One example of such a graphical form of data representation is a data model known as Resource Description Framework (RDF), which represents data in the form of a mathematical graph, that is to say a graph of nodes and directed arcs, and in doing so illustrates any interrelationship of different data attributes. In accordance with the terminology of the RDF data model, data is represented either as a Resource, a Property, or a Value. One of the values of representing data graphically is that, in theory it is possible to allow data thus represented to convey semantic meaning, and for this reason RDF is currently the leading candidate data model for providing the basis of a semantic Worldwide Web. An example of the use of RDF is illustrated in Figs. 1 and 2, which show a tabular representation of two conventional database entries, and the representation of the data forming the entries of Fig. 1 in RDF respectively.

Referring now to Fig. 1, two records whose data it is desired to store in a database are illustrated. Each record has three attributes: the publication number of a patent, the inventor designated on the patent, and the author of the specification of the patent. As can be seen from looking at the records, the inventor in each case is the same, and so to this extent at least, the two records are interrelated.

Referring now to Fig. 2, both records, and their interrelationship can be represented in a graphical document format known as Resource Description Framework (RDF), and an RDF document representative of the two records is shown in Fig. 2. The RDF document may be thought of as graphical representation of the data in Fig. 1, which also describes the structure of that data, and contains essentially three elements: Resources, Properties and Values. Thus for example, the document in Fig. 2 has a resource #A1. This Resource is labelled #A1, although in the event that the resource could be named by a Uniform Resource Indicator (URI), such as for example a web page address, this would also appear in the name of the Resource. In this example the resource has no such name, but has four different properties which, *inter alia* serve to characterise it: Patent No., Author, Inventor (all of which may intuitively be related to one of the records in Fig. 1), and "rdf: type". The first three properties are simply the different attributes of one of the records shown in Fig. 1, while the fourth indicates the type or nature of the Resource, which in this instance is a patent. With this in mind it follows that a patent (which is the "type" of the Resource) has the properties of Author, Inventor and Number, and while this may not be the most intuitive way to describe a record in Fig. 1 from a lay person's perspective, it nonetheless is possible to see that all of the information shown in a record in Fig. 1 is replicated in this format. Thus the two Resources #A1 and #B1 relate to the patents 5678 and 1234 respectively.

The properties of Inventor and Author for each of these two Resources are respectively represented by further Resources: #B2 which corresponds to the inventor – since the inventor is the same in each case; and #A2 and #C2 which correspond to the two authors. The Resource #B2 is thus the Value of the Inventor Property for each of the Resources #A1 and #B1, and itself has two further properties, one of which is its rdfs: type, indicating that the Inventor is a person, and the other is the name of the inventor, which is its "literal" Value, the inventor's name A. Dingley. The Author Properties of the Resources #A1 and #B1 are respectively the Resources #A2 and #B2 and each have an rdfs: type property which signifies that the Author is a person, and Name Properties having literal Values, which are the names of the Authors "Formaggio" and "Cheeseman" respectively.

Thus an RDF document describes completely both the data in a record, its nature and any interrelationship with data in another record. The purpose of representing data in such a manner is essentially to provide a common format independent of the source format of data, which may be manipulated by computers, and which contains all of the original data.

One of the problems associated with the graphical representation of data, which is a problem well known *per se,* is that limitations of current mathematical theory correspondingly limit the ability to process the data. For example, it is not within the scope of current mathematical theory to provide, analytically, a rigourous topographical comparison between two graphs. Such limitations in turn limit the extent to which graphically represented data can achieve the aims of providing the basis of, for example, a semantic worldwide web.

A first aspect of the present invention seeks, *inter alia* to ameliorate this problem, and provides a generally applicable method of processing data, which is applicable to the processing of graphically represented data. According to a first aspect of the invention there is provided a method of processing data (typically, but not necessarily graphical data) according to which data is processed in accordance with a first set of rules, which operate, *inter alia* to define a stage at which such a processing operation ceases, and then applying to the partly-processed data a second set of rules, which operate to modify the data so that the data thus modified is then processable further by applying a third set of rules. In a preferred embodiment, modified data is then processed by the third set of rules.

Although the data which it is desired to process has itself been changed, because this change has taken place on the basis of a defined set of rules the outcome of these operations, and in particular the manner in which the outcome of the processing differs from an ideal (in which the unmodified data is processed completely), will be well understood. Different sets of data processed by applying this method may thus be compared, combined or otherwise used in conjunction with each other on this basis provided consistent rules are applied to their processing. In one preferred embodiment, the first and third sets of rules are similar, and in a further embodiment they are the same.

Preferably the method outlined above is preceded by the deterministic modification of the data prior to applying the first set of rules. Deterministic modifications may, in layman's terms, be thought of as modifications which are not significant. Thus by analogy, the colour of the paper on which a patent specification is printed is not significant *vis a vis* it's legal effect, and may thus be thought of as being insignificant. Conversely, non-deterministic modifications are modifications which *are* significant.

Preferably modification of the data following processing in accordance with a first set of rules is/are non-deterministic modifications (i.e. significant), but may be "labelled" as insignificant.

A more detailed description of an embodiment of the present invention will now be provided. Firstly in a prototype implementation of the method for UNIX, shown below, and secondly in the accompanying proposal for a conference paper entitled "RDF Canonicalization A Cheater's Guide".

APPENDIX – Unix and GNU awk (version 3.0.3) partial implementation

```
=========================
== README
=========================


This directory contains a prototype implementation
of the techniques described in RDF Canonicalization:
A Cheater's Guide by Jeremy J. Carroll.

The implementation requires Ntriple files as input
and outputs Ntriple files in canonical RDF.

The treatment of space in literals does not
conform with the lexicographic order in the paper;
spaces are sorted as the character sequence "\u0020".

The files are:
 aline       - A file with two or more blank lines.
 c14n.awk     - An awk script implementing steps 7 and 8
             from the one step deterministic labelling.
 escape-space.sed
            - a sed file for removing spaces from literals
             in Ntriple files. (They are replaced with
             the illegal escape sequence \u0020)
multipass.awk - An awk library to allow multiple passes
             over the datastream in a single awk process.
multistep.awk - An awk script that initializes a one or multi
             step deterministic labelling, using the
            c14n.awk and multipass.awk libraries.
multistep.sh  - A shell script that invokes the sed and awk
             scripts appropriately - start here.
multistepdelete.sh - A shell script that turns arbitrary
             Ntriples into canonical RDF by deleting
             the unlabelled nodes after a multistep
             labelling.


=========================
== aline
=========================
```

= c14n.awk

```


# This file defines the following awk passes that
# can be used with the multipass.awk library.
# "step7" in the one-step  deterministic labelling.
# "step8" in the one-step  deterministic labelling.

 { dType = 0
  dSame = 0
 }
 $1 == "~" { dType += 1 }
 $3 == "~" { dType += 2 }
 PASS == "step4" && $1~/^_:/ { $(NF+1) = "#"
                          $(NF+2) = $1
                          $1 = "~"
                         }
 PASS == "step5" && $3~/^_:/ { $(NF+1) = "#"
                          $(NF+2) = $3
                          $3 = "~"
                         }
 PASS == "step4" { print > tmp }
 PASS == "step5" { print > tmp }
 PASS == "deleteSome" && ( $1 == "~" || $3 == "~" ) { print > tmp }
 PASS == "step7" && FNR != 1  && dType != 0 {
   dSame = dLastType!=0 && ( dLast[1] == $1 && dLast[2] == $2 && dLast[3] == $3 )
 }
 PASS == "step7" && FNR != 1  && dLastType != 0 {
   if( (!dSame) && (!dLastSame) ) {
     numberVars()
   }
 }
 PASS=="step7" {
   printLast()
   dLastSame = dSame
   dLastType = dType
   dLastLength = split($0,dLast)
  }
  PASS=="step8" { useNumberVars(); print > tmp }
  AFTERPASS=="step7" {
    if ( dLastType != 0 && (!dLastSame) ) {
     numberVars()
    }
    printLast()
    dLastLength = 0
   }

   function printLast(   di)
  {
    if ( dLastLength != 0 ) {
   for ( di = 1 ; di < dLastLength ; di ++ ) {
     printf "%s ",dLast[di] > tmp
    }
    printf "%s\n",dLast[di] > tmp
    }
  }

  function numberVars() {
    numberVar(3)
    numberVar(1)
   }
   function numberVar(ix) {
    if ( dLast[ix]=="~") {
     name = dLast[dLastLength]
     dLastLength--
     if ( !dTable[name] ) {
      dTable[name] = sprintf("%6.6d",counter++)
     }
     dLast[ix] = "_:g" dTable[name]
     if ( dLast[dLastLength] != "#" ) {
      print "Shouldn't happen" > "/dev/stderr"
     } else {
      dLastLength--
     }
    }
   }
```

```
function useNumberVars() {
 useNumberVar(3,NF)
 useNumberVar(1,6)
}
function useNumberVar(ix,pos) {
  if ( $ix =="~") {
   if ( !dTable[$pos] ) {
     return
   }
    $ix = "_:g" dTable[$pos]
   if( pos == NF ) {
     NF -= 2
   } else {
     $pos = $(pos+2)
     NF -= 2
   }
  }
 }
==============================
== escape-space.sed
==============================
      :retry
      #s/\(a[^\\]"\)b/\1\1/
      /^#/d
       s/\(^[^"]*[^\\]"\([^\\"]\|\\.\)*\) /\1\\u0020/
      #s/\([^\\]"\([^\\"]\|\\.\)*\) /\1\\u0020/
      t retry
==============================
== fmtx
==============================
      #!/bin/csh
      foreach i ( $* )
        echo "=============================="
        echo "== " $i
        echo "=============================="
         pr -t -n $i
      end

==============================
== multipass.awk
==============================
       # This is a library that allows the use of awk in
# multipass mode over stdin.
# use addPass("foo") to add an extra pass over
# the input. The pass is named "foo".
# To print output for use in the next pass
# use print > tmp
# To access the name of the current pass
# use the variable PASS
# A special pass called "sort" maybe added,
# this sorts the data in lexicographic sort order


BEGIN {
      tmp = "tmpA"
      otherTmp = "tmpB"
       ARGC = 1
    }

 { if ( DEBUG ) print FILENAME, ARGIND, passNames[ARGIND], $0 > "/dev/stderr" }
 { AFTERPASS = 0
  if ( FNR == 1 && ARGIND % 2 == 0 ) {
     AFTERPASS=PASS
  }
 }
 { if ( FNR == 2 && ARGIND % 2 == 0 ) {
     if ( passNames[ARGIND-1]=="sort" ) close(sort)
     else close(tmp)
     x = tmp
     tmp = otherTmp
     otherTmp = x
    nextfile
  }
 }
 { if ( FNR == 1 ) {
    PASS  = passNames[ARGIND]
   if ( ARGC == ARGIND+2 ) tmp = "/dev/stdout"
    if ( PASS == "sort" ) {
     if ( ARGC == ARGIND+2 ) sort = "sort"
     else sort = "sort > " tmp
   }
  }
}

```

```
PASS == "sort"  { print | sort }

function addPass(passName,   i)
{ i = (ARGC + 1)/2
  passNames[ARGC] = passName
  ARGV[ARGC] = ( i % 2 == 0 ? "tmpA" : "tmpB" )
  if (ARGC==1) ARGV[1]="/dev/stdin"
  ARGV[ARGC+1] = "aline"
  ARGC += 2
}





```

```
== multistep.awk
```

```

 # This implements the multistep and the onestep
# deterministic labelling algorithm from the
# paper RDF Canonicalization: A Cheater's Guide
# by J. Carroll.

 # The command line should be:
# awk -f multipass.awk -f c14n.awk -f multistep.awk [NN] < ntriples.nt
# [NN] is the desired number of steps (default one)
 # ntriples.nt is the RDF graph as an ntriples file as input.
# Note  spaces in literals are not supported.


BEGIN {
  if (ARGC == 0) STEPS = 1
  else STEPS = ARGV[ARGC]
  addPass("step4")
  addPass("step5")
  addPass("deleteSome")
   addPass("sort")  # step6
   for ( i = 0; i<STEPS; i++) {
    addPass("step7")
    addPass("step8")
    addPass("sort")
```

```
        }
    }
```

================================
== multistep.sh
================================

```
#!/bin/sh
export LC_ALL=C
sed -f escape-space.sed | awk -f multipass.awk -f c14n.awk -f multistep.awk $1
```

================================
== multistepdelete.sh
================================

```
#!/bin/sh
export LC_ALL=C
./multistep.sh $1 | sed -e '/~/d'
```

RDF Canonicalization A Cheater's Guide

Jeremy J. Carroll[1]
Hewlett-Packard Labs
Bristol
UK. BS34 12QZ

jjc@hpl.hp.com

ABSTRACT.

Digital signatures require canonical forms for the data structures being signed. Given the importance of both the Resource Description Framework (RDF) and digital signatures in semantic web architecture, it is highly desirable to have polynomial time canonicalization algorithms for RDF. This paper demonstrates that RDF canonicalization is Graph Isomorphism complete, and hence that, probably, no such polynomial time algorithm exists. However, a practical solution that provides near-canonicalization of any RDF graph in subquadratic time is demonstrated. This solution relies on early identification of the difficult blank nodes and then adding marker triples; slightly changing the RDF graph into a straightforwardly canonicalizable one.

Categories and Subject Descriptors

G.2.2 [**Graph Theory**]: *Graph algorithms, Graph labeling*. F.2.2 [**Nonnumerical Algorithms and Problems**]: *Pattern matching, Complexity of proof procedures, Computations on discrete structures, Sorting and searching* I.7.2 [**Document Preparation**]: *Markup languages, Standards*

General Terms

Algorithms, Standardization

Keywords

RDF, digital signatures, XML canonicalization, OWL, graph isomorphism.

1. INTRODUCTION

This paper concerns techniques to canonicalize RDF graphs. In other words, a method of choosing one of the very many different ways of writing down an RDF graph.

In many technical areas we find equivalences between things. This results in two superficially different objects being regarded as the same. Canonicalization is an important technique to side step some of the less desirable consequences of the superficial differences.

Two types of canonicalization appearing in web technologies are the canonical lexical representatives of specific datatype values in XML Schema Datatypes [6], and the canonicalization of XML documents and document subsets [7].

[1] Jeremy Carroll is a visiting researcher at ISTI, CNR Pisa.

Canonicalization within web technologies is largely concerned with lexical considerations.

One of the crucial applications that XML Canonicalization [7] is intended to service is that of digital signatures. Digital signature are applied to sequences of bytes; and can be used to detect changes in such. Without a canonical form it is necessary to store the original XML document along with the representation used during further XML processing. A further advantage that the XML Canonicalization recommendations bring is the ability to express a document subset as a well-defined byte sequence that can be signed, independently of the full document.

1.1 Semantic Web Architecture

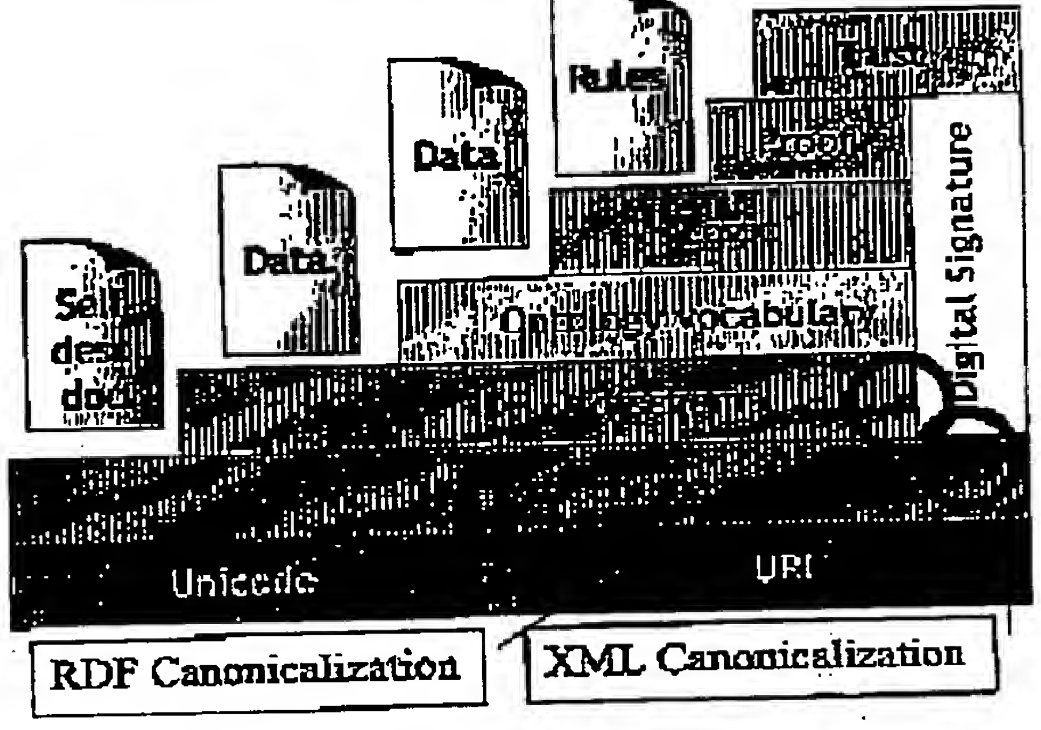

Figure 1: Berners-Lee's Architecture [3]

For the semantic web [5] and semantic web web services [14], XML documents serve merely as a transport for a deeper meaning, that is expressed in terms of RDF graphs. Digital signatures play a crucial role in enabling the trust layer to be built on top of the lower layers. In Figure 1, Berners-Lee's well-known architecture picture is annotated to show the position both of the XML Canonicalization recommendations, and the work of this paper.

Canonicalizing the semantic layers of the web at the level of the XML documents would be as misplaced as canonicalizing XML documents only at the level of character encoding.

1.2 XML C14N

In XML there are a number of arbitrary choices made which are not generally seen as contributing to the 'meaning' of the document. While it is unclear quite what that meaning might be, none of the XML family of recommendations see certain aspects of an XML document as significant. These include the choice of character encoding; the choice between single and double quotes for attribute values; and the choice of white space within element tags. Canonical XML [7] determines preferred options for such choices, and, in fact, for all aspects of the document that are insignificant in the XPath nodeset [11].

These choices may change how the document looks in a simple text editor.

XML canonicalization is the process by which an XML document is converted into another one, which is in canonical XML, but is otherwise identical to the original.

1.3 Canonical Ordering for RDF

In XML, the document ordering is largely preserved during canonicalization. Whenever two elements of the XML Information set appear in the canonicalized document, they appear in the same relative order as they were found in the precanonicalized form.

RDF does not have a notion corresponding to XML's document ordering. The RDF abstract syntax [18] is defined as an (unordered) set of triples.

Any specific document containing a serialization of an RDF graph will however have imposed some total order on those triples.

Thus any canonicalization of RDF involves a canonical ordering of the triples in the RDF graph. Almost all of this paper deals with that part of the problem; the details of how, given such an ordering, the file can actually be written are sketched quickly.

1.4 Difficulties with RDF Canonicalization

The fundamental problem is that RDF Graph canonicalization can be shown to be equivalent to the Graph Isomorphism problem. The complexity (GI) of this latter problem is well-researched [20] and is conjectured to be strictly harder than polynomial time, and strictly easier than non-deterministic polynomial time. i.e.

$$P < GI < NP$$

Such a high complexity is unacceptable in an infrastructural component.

1.5 Meaningless Changes

The key insight of this paper is that while in general RDF Graph canonicalization is GI complete, all interesting RDF graphs can be slightly modified (typically by adding a few, explicitly meaningless, arcs) to be in a class of RDF graphs which can be much more easily canonicalized ($O(n\log n)$). The reader should judge whether these modifications are merely a dirty hack or an elegant engineering compromise.

An alternative engineering approach of simply deleting problematic parts of the graph is also explored.

2. A SIMPLE EXAMPLE

Consider an RDF Graph with two triples. It can be represented in N-Triples [16] as a file:

```
        # Here is a graph
        _:aBlankNode <eg:prop>      "3" .
        _:aBlankNode    <eg:prop> "5"    .
```

It can also be represented as:

```
        _:ax       <eg:prop> "5"    .
        _:ax  <eg:prop> "3" .
```

The two files are different, but the graph they describe is the same. The differences between the files are *insignificant*. These include:

- The comments.
- The whitespace.
- The order of the lines.
- The choice of blank node identifiers (e.g. *aBlankNode* or *ax*)

Other aspects of the N-Triples are *significant* in that they represent the intended abstract RDF Graph [18]. Significant aspects include:

- The presence or absence of a triple
- The string in each literal
- The URIs for each property or resource

Canonical RDF is chosen so that there is at most one representation of that graph in Canonical RDF. Thus, we can check that these two are the same by converting both into canonical RDF and doing a character-by-character comparison.

Of the insignificant differences highlighted, only two present any interesting difficulties: the order of the lines, and the blank node identifiers.

3. APPLICATIONS OF RDF C14N

The techniques described in this paper can be used to enable various applications over semantic web technologies. These correspond to the applications enabled by XML Canonicalization [7].

There are two applications scenarios made explicit in that recommendation:

- testing whether the information content of an XML document or document subset has been changed
- digital signature generation over the canonical form of an XML document

Lynch [21] argues that canonicalization is a "Fundamental Tool to Facilitate Preservation and Management of Digital Information." The scenario he discusses is the need to reformat a digital object which is being preserved in an electronic library. The need for reformatting occurs as the software and hardware required to view the original object become obsolete.

Lynch uses (lossy) canonicalization as a way of defining the essence of a document. This essence, if sufficient unimportant details are discarded, will be invariant with reformatting. Thus we can compute this essence of the original document, and the original author can sign this essence to vouch for its authenticity.

Later, after the death of the author, and a sequence of many transformations of the original byte sequence, the essence of the document is unchanged, and we can still verify that the author signed it.

Within the semantic web, a particularly important application of RDF canonicalization is likely to be the signing of OWL ontologies [13].

4. RDF C14N IS GI COMPLETE

The techniques in this paper cannot canonicalize arbitrary RDF graphs without making some modifications to them. This reflects the underlying difficulty of RDF canonicalization.

As discussed by Carroll [8], RDF graph equality and the graph isomorphism problem have equivalent complexity.

Any unlabelled undirected graph can be encoded in RDF by replacing each node with a blank node, and replacing each arc by two arcs (in each direction) always using a single property (e.g. <eg:a>). So a simple triangle (three nodes, three arcs) can be encoded in N-Triples as:

```
_:a <eg:x> _:b .
_:a <eg:x> _:c .
_:b <eg:x> _:a .
_:b <eg:x> _:c .
_:c <eg:x> _:b .
_:c <eg:x> _:a .
```

If we could solve RDF canonicalization in polynomial time, then we could compare two RDF graphs for equality in polynomial time (by comparing their canonical representations). This would provide an (unlikely) polynomial time solution to the Graph Isomorphism problem [15], [20].

4.1 What's the difficulty?

The graph isomorphism problem is deceptive. It really doesn't look hard! We will informally explore the problem of a canonical representation of a simple graph, with two disconnected components shown in figures 2 and 3.

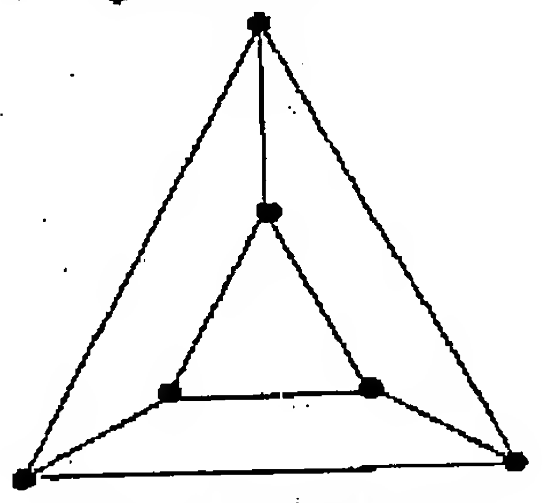

Figure 2 A 6 vertex graph

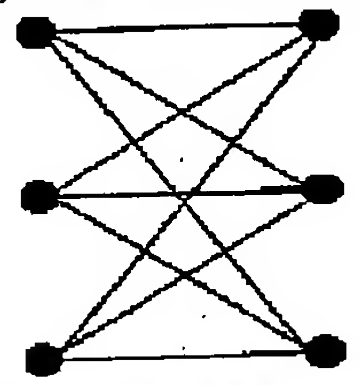

Figure 3 A different 6 vertex graph: $K_{3,3}$

The graph consists of 12 nodes and 18 edges. Each node has three neighbours, and two nodes at a distance of two away. Edge node is in a connected component of six nodes.

If we try and canonicalize this, we will start by writing some node before all the other nodes. That node will either come from the component in figure 2 or that in figure 3. The canonicalization algorithm needs to make a deterministic choice. There is no intuitively obvious rationale for choosing one or the other, and which ever choice we make seems to require considerations not just about the node, but also its neighbours, and their neighbours, and . . .

A good solution to this problem is provided by McKay [22], [23]. He uses an analysis of the automorphism group of a graph to construct canonical representatives. His solution is of non-polynomial complexity and complicated to program. It is not a plausible candidate for an infrastructural component within the semantic web.

We suggest other techniques, described below, which when presented with the rather unlikely RDF that encodes such an unlabelled undirected graph, will quickly work out that there is a problem. One of the methods canonicalizes only that part of the graph which is not problematic: that method will delete all the edges in this example. The other method will change the example by randomly adding enough labelled nodes and directed edges to make it possible to choose which node goes first based on lexicographic orderings of the labels.

These methods are successful for RDF because real RDF data is not as difficult as these examples. Hence, the rather arbitrary and unwelcome aspects of both methods, in practice only get used a little. For example, rather than the catastrophic change of deleting all the edges, in practice, that method deletes only a small percentage of the edges.

5. PRELIMINARIES

5.1 Syntactic

For clarity of exposition the algorithms described in this paper are based around the lexicographic sorting of N-Triples [16] files. N-Triples is an ASCII format.

We use numbered gensym identifiers during the algorithm. These are created with just sufficient leading zeros so that lexicographic ordering and integer ordering are the same. (The number of leading zeros required is computed from the number of blank nodes in the RDF graph being considered).

5.2 Semantic

The techniques in this paper rely on being able to make meaningless changes to an RDF graph. This is done in accordance with the RDF formal semantics [17], by using a special property, which we conventionally refer to as `c14n:true` defined here:

```
<rdf:RDF
   xml:base="&c14n;"
   xmlns:c14n="&c14n;#"
   >
  <rdfs:Property rdf:ID="true">
     <rdfs:description>
  This property is true whatever resource is
its subject, and whatever literal is its
object.
   Thus triples with literal objects, and
c14n:true as predicate, can arbitrarily be
added to and deleted from an RDF graph
without changing its meaning.
     </rdfs:description>
  </rdfs:Property>
<rdf:RDF>
```

The entity reference `&c14n;` is bound to the URL `http://www-uk.hpl.hp.com/people/jjc/rdf/c14n` .

The entity declaration and the declarations of the `rdf` and `rdfs` namespaces have been omitted.

By specifying this predicate as being always true, adding or deleting triples with this predicate does not alter the entailments under the RDF model theory. Formally the semantics of the document are unchanged.

5.3 N-Triples to Canonical XML

In this paper, we concentrate on creating canonical representations of RDF graphs in N-Triples [16].

However, for full integration with tools built on Canonical XML [7] it is necessary to transform these files into XML, and, moreover, we wish the XML produced to canonically depend on the original RDF graph.

Conventionally we use `rdf` as the prefix for the RDF namespace declared on the root element of the document.

For simplicity we will turn each triple in an N-Triples document into an `rdf:Description` element, which contains a single property element.

If the subject of the triple is a URI reference, then we use an `rdf:about` attribute on the `rdf:Description` element, otherwise the subject is a blank node and we use an `rdf:nodeID` attribute. We use the blank node identifier from the N-Triple as the blank node identifier in the RDF/XML.

The predicate of the triple is expressed using the default namespace on the property element, with the predicate URIref being split at the leftmost legal point. I.e. the local name is as long as possible. (Some predicates have no legal split point and such graphs cannot be serialized in RDF/XML, see [2]).

If the object of the triple is a URI reference, then we use an `rdf:resource` attribute on the property element. If the object is a blank node, we use an `rdf:nodeID` attribute. If the object is an XML Literal we use `rdf:parseType="Literal"`. If the object is a typed literal we use an `rdf:datatype` attribute. For all literals we put the lexical form into the element content.

We use a newline after the opening `<rdf:RDF>` tag, and after each property element end tag. Otherwise we only use whitespace within element tags and as specified by literal lexical forms.

The triples are converted into XML in order.

The resulting XML file is then canonicalized [7].

6. RDF C14N WITHOUT BLANK NODES

To create a canonical N-Triples file for an RDF graph without any blank nodes we use the following algorithm:

1. Canonicalize each XML Literal [18] in the graph using XML canonicalization [7].
2. For each typed literal in the graph canonicalize[2] it according to the rules in XML Schema datatypes [6]. That is, given a typed literal `<datatypeURI, lexicalForm>` replace it with `<datatypeURI, lexicalForm'>`, where `lexicalForm'` is the canonical from of `lexicalForm` according to the datatype specified by `datatypeURI`.
3. Write the graph as an N-Triples document [16]. Each line of the document is a complete distinct triple of the graph.
4. Reorder the lines in the N-Triples document to be in lexicographic order. (This could be implemented simply with Unix™ `sort`[3]).

[2] Datatypes that are not XML Schema built-in types are not supported in this version.

[3] `sort` in Unix uses a locale dependent ordering. To use US-ASCII order, it is necessary to set the environment variable LC_ALL=C.

7. LABELLING BLANK NODES

If we have blank nodes in the graph then life is somewhat trickier.

In N-Triples blank nodes are represented using blank node identifiers, which can appear in subject or object position.

Unfortunately, these identifiers are gensyms created during the writing of the N-Triples, and are not an intrinsic part of the graph. Hence, it is an error if the canonicalization depends on these gensyms. In contrast, the canonicalization algorithm must deterministically choose new blank node identifiers.

In this part of our approach, we first write out the file using arbitrarily chosen blank node identifiers; then we sort the document (mostly) ignoring those identifiers. On the basis of this sorted document, we then rename all the blank nodes, in a (hopefully) deterministic fashion.

Since the level of determinism is crucial to the workings of the canonicalization algorithm, we start by defining a deterministic blank node labelling algorithm. This suffers from the defect of not necessarily labeling all the blank nodes.

Deterministic, in this context, means dependent only upon significant parts of the initial representation of the graph, and *nondeterministic* means dependent, in part, upon insignificant parts of the initial representation of the graph.

7.1 One-step Deterministic Labelling

In RDF [18], a single triple can contain zero, one or two blank nodes (the property must be specified by a URI [4]).

We present an algorithm that labels some blank nodes on the basis of the immediate neighbors of that blank node (the one-step in the name of the algorithm).

The algorithm from section 0 is modified:

1. Canonicalize each XML Literal [18] in the graph using XML canonicalization [7].
2. For each typed literal in the graph canonicalize it according to the rules in XML Schema datatypes [6].
3. Write the graph as an N-Triples document. Each line of the document is a complete distinct triple of the graph.
4. For each line with a blank node identifier in subject position (e.g. `_:subj`), replace the blank node identifier with "~" and add a comment "# `_:subj`" to the end of the line, indicating the original identifier.
5. For each line with a blank node identifier in object position (e.g. `_:obj`), replace the blank node identifier with "~" and add a comment "# `_:obj`" to the end of the line, indicating the original identifier.
6. Reorder the lines in the N-Triples document to be in lexicographic order. (This could be implemented with Unix™ `sort`).
7. Use a gensym counter, initialized to 1, and a lookup table, initially empty. Go through the file from top to bottom:
   a. If this line is the same as the next or previous line excluding any trailing comment, continue to the next line in the file.
   b. If there is a "~" in object position:
      i. Extract the blank node identifier from the final comment in the line. Remove the comment.

ii. Look the identifier up in the table.

iii. If there is no entry, insert a new entry formed from "_:g" concatenated with the current gensym counter value. Increment the counter.

iv. Replace the "_" with the value from the table.

c. If there is a "_" in subject position, use the same subprocedure to replace it with a consistently chosen gensym.

8. Using the same lookup table. Go through the file from top to bottom:

a. If there is a "_" in object position:

i. Extract the blank node identifier from the final comment in the line. Remove the comment.

ii. Look the identifier up in the table.

iii. If there is an entry, replace the "_" with the value from the table.

b. If there is a "_" in subject position, use the same subprocedure to possibly replace it with a consistently chosen gensym.

9. Lexicographically sort the N-Triples again.

The only non-determinism here is that the sort may depend on the blank node labels in pairs of triples for which the rest of the triple is identical. Such pairs are studiously avoided in the assignment of labels, and so the order in which such pairs appear does not effect the labels chosen. Notice that step 8, which does deal with the incomparable lines, does not choose any labels and is hence deterministic.

The algorithm will deterministically label some of the blank nodes, for others it leaves them unlabelled. These nodes are referred to as *hard to label* nodes.

The operation of the deterministic labeling algorithm depends on triples that can distinguish one blank node from another. These *distinctive triples* are characterized by being unique in the graph even when all blank nodes are treated as identical.

The hard to label nodes do not participate in any distinctive triples.

Example 1, is fully labeled by this algorithm:

```
<eg:a> <eg:foo> _:a .
_:a <eg:prop> "val" .
<eg:b> <eg:prop> _:b .
```

are transformed into these:

```
<eg:a> <eg:foo> _:g1 .
<eg:b> <eg:prop> _:g2 .
_:g1 <eg:prop> "val" .
```

On the other hand, example 2 is not fully labelled:

```
_:a <eg:zee> "why" .
_:a <eg:prop> "val" .
<eg:b> <eg:prop> _:b .
_:b3 <eg:prop> "val" .
```

are transformed into these:

```
<eg:b> <eg:prop> _:g1 .
_:g2 <eg:prop> "val" .
_:g2 <eg:zee> "why" .
_ <eg:prop> "val" . # _:b3
```

8. CANONICAL RDF

The algorithm above has the desired property of being of a low complexity class, and hence optimizable to be sufficiently quick.

Thus we will define canonical RDF on the basis of this algorithm.

A canonical N-Triples document is one (without any comments) which is unchanged under the application of the one-step deterministic labeling algorithm.

That is canonical RDF in N-Triples has the following features:

- There are no hard to label nodes.
- Every blank node identifier has the form gNNN where NNN is some number of digits. The number of the digits is the same for every blank node identifier. At least one identifier has a non-zero first digit.
- After deleting all triples which are not distinctive, the first occurrences of each blank node identifier appear in numeric order, starting at 1, without gaps.
- The file is in lexicographic sort order.

Such files are unchanged under the one-step deterministic labeling.

If one-step deterministic labeling successfully labels all the nodes, then the resulting output will be canonical RDF. In such cases, one-step deterministic labeling is idempotent. So the resulting file in example 1 above is in canonical RDF.

9. THE CHEATER'S GUIDE

The one-step deterministic labelling algorithm is sufficiently quick, being O(nlogn), and the goal of the rest of the paper is to make such an algorithm useable in all cases: i.e. instead of looking for an algorithm that completely solves the RDF graph canonicalization problem, we will modify the problem to be soluble by the algorithm: we will ensure that there are no hard to label nodes. We show two ways of modifying the graph to make a deterministic labeling algorithm work.

This process of making significant and/or nondeterministic modifications to the input in order to make it easier to canonicalize is not acceptable in a true canonicalization algorithm. Thus we refer to it as 'cheating'; we try to cheat only a little bit; and to be informed by the application needs as to what sort of cheating we use.

9.1 Deterministic Cheating

The simpler approach is to delete all hard to label nodes from the graph. This involves deleting all the triples in which such nodes participate.

None of those deleted triples will be distinctive, because if they were, then the blank nodes in them would have been labeled.

The modified algorithm is:

A. Perform the one step deterministic labeling.

B. Delete all lines containing a "_" in subject or object position.

We note that this procedure is deterministic and idempotent.

Deleting unlabelled nodes appears drastic, but as long as there are few (e.g. zero) of these nodes, it is a good practical solution.

Continuing example 2 we have:

```
<eg:b> <eg:prop> _:g1 .
_:g2 <eg:prop> "val" .
_:g2 <eg:zee> "why" .
```

after the application of this algorithm.

For applications such as Lynch's desire to capture the essence of a digital object despite reformatting this is a good fit. [21]. This allows verifying the long-term integrity of digital deposits, where the entire meaning does not need to be canonicalized, as long as the greater part is captured in the canonicalization.

9.2 Non-deterministic Cheating

Other applications of the use of digital signature require that the meaning of the document is fully captured in the object that is signed. A (somewhat tired) example would be signing a purchase order. Hence the deterministic approach above is unsatisfactory, since it quite happily changes the meaning of a document.

The second approach to cheating adds distinctive triples in order to make sure that none of the nodes are hard to label.

The version described here uses many passes of the file – the three important passes:

- identify the hard to label nodes;
- create additional distinctive triples for those nodes
- deterministically label the resulting graph.

In the preliminaries we arranged a ready supply of meaningless triples (with predicate c14n:true) that we can add to and delete from the graph without modifying its meaning.

Thus, we can perform the following steps:

A. Perform a one-step deterministic labeling

B. If there are no hard to label nodes, then stop.

   [This step is to ensure the algorithm is idempotent]

C. Delete all triples with predicate **c14n:true** .

   [Without B this would not be idempotent]

D. Perform a one-step deterministic labeling.

E. Using a new lookup table from step D, and a new counter, scan the file from top to bottom, performing these steps on each line:

   a. If there is a "~" in object position:

      i. Extract the blank node identifier from the final comment in the line. Remove the comment.

      ii. Look the identifier up in the table.

      iii. If there is no entry: add an entry "x" to the table; and add a new triple to the graph with subject being the blank node identified by the identifier from the comment, predicate being **c14n:true** and object being the string form of the counter; increment the counter.

   b. If there is a "~" in subject position, use the same subprocedure to possibly create a distinctive triple for the subject as well.

F. Perform a one-step deterministic labeling (of the new modified graph, with a new lookup table and counter). Since all nodes participate in a distinctive triple, every table lookup will find an entry.

This cheating is non-deterministic in step E, but that non-determinism is fairly limited, because even with the rather naïve one-step deterministic labeling algorithm almost all nodes in almost all (practically occurring) RDF graphs will have been classified.

So again continuing with example 2, we find:

```
<eg:b> <eg:prop> _:g1 .
_:g2 <eg:prop> "val" .
_:g2 <eg:zee> "why" .
_:g3 <eg:prop> "val" .
_:g3 <http://www-uk.hpl.hp.com/people/jjc/rdf/c14n#true> "1" .
```

An application scenario in which this is useable is that of signing an OWL ontology [13]. The ontologist creates the ontology in a tool, and then asks the tool to generate a signature using a private key. The tool:

- applies this algorithm, possibly adding additional triples to the ontology.
- creates a canonical RDF graph with the same meaning as the original ontology.
- computes the signature for the canonical RDF graph.
- adds additional triple(s) to the graph with the ontology URI as subject, and the signature as object.

The resulting graph (with additional triples both as a result of canonicalization and reflecting the signature) then replaces the original ontology. It is this graph (which can be canonicalized without any changes) that the ontologist publishes. Users of this ontology can then:

- find the signature in the graph.
- find the public key using some public key infrastructure
- delete the triple(s) carrying the signature from the graph.
- apply the deterministic labeling to form a canonical representation of the graph
- verify the signature of this canonical RDF using the public key.

9.3 Further Discussion

The one-step deterministic labeling algorithm used had the following characteristics:

- Reasonably fast (subquadratic)
- Deterministic
- Labels 'enough' blank nodes in realistic RDF
- Easy to explain and implement

The last point is helpful in a paper, but not a hard requirement. Thus in a deployed system we can imagine using a variant of the algorithm here.

It is a practical engineering problem to choose a near-optimal deterministic classification algorithm that gets the best trade off between speed of classification and practical utility. The deterministic labeling algorithm used in this paper is probably a touch naïve, but only a touch. We can be sure that those used by Mackay [22] are too expensive, given their non-polynomial complexity.

10. ADVANCED METHODS

10.1 Multistep Deterministic Labelling

The one-step deterministic labeling only considers the immediate neighbours of any node. In some graphs, particularly those arising in OWL ontology definitions [13], this may leave too many nodes unlabelled.

The solution is to choose a fixed depth e.g. two or three, and to consider the neighbours of a node up to that distance away. Thus we can consider the two-step and three-step deterministic labeling algorithms. Since for a fixed k a k-step method will, in essence, be k applications of the one step method, we will expect a linear slow down as we increase k. The underlying complexity class remains O(nlogn). (It is straightforward to optimize k-step methods to make the slow down much less than linear).

To consider neighbours at distance N+1 we resort the partial labeled output of a deterministic labeling based on neighbours at distance N.

Having resorted the output, which will take into account those labels already deterministically chosen, we can then apply the one pass deterministic labeling algorithm, which effectively considers neighbours at an additional step removed.

For example, the full two-step deterministic labeling is as follows:

A. Perform a one step deterministic labeling.

B. Repeat steps 7, 8, 9 from the one pass deterministic labeling algorithm, without reinitializing the table or counter.

For a k-step labeling, we need to repeat B k-1 times.

Example 3 is a simple OWL expression, written in RDF/XML as:

```
<rdf:RDF xmlns:owl = "http://www.w3.org/2002/7/owl#"
  xmlns:eg="eg:"
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
   <owl:Class>
     <owl:oneOf rdf:parseType="Collection">
       <rdf:Description eg:sugar="Dry" />
       <rdf:Description eg:sugar="OffDry" />
       <rdf:Description eg:sugar="Sweet" />
     </owl:oneOf>
   </owl:Class>
</rdf:RDF>
```

The corresponding triples (using qnames as in the OWL Test Cases [9]) are:

```
_:j0 rdf:type owl:Class .
_:j2 <eg:sugar> "Dry" .
_:j4 <eg:sugar> "OffDry" .
_:j6 <eg:sugar> "Sweet" .
_:j5 rdf:first _:j6 .
_:j5 rdf:rest rdf:nil .
_:j5 rdf:type rdf:List .
_:j3 rdf:first _:j4 .
_:j3 rdf:rest _:j5 .
_:j3 rdf:type rdf:List .
_:j1 rdf:first _:j2 .
_:j1 rdf:rest _:j3 .
_:j1 rdf:type rdf:List .
_:j0 owl:oneOf _:j1 .
```

Applying the one-step deterministic labeling fails to label one of the nodes (still using qname syntax for URIs):

```
_:g0 <eg:sugar> "Dry" .
_:g1 <eg:sugar> "OffDry" .
_:g2 <eg:sugar> "Sweet" .
_:g3 rdf:first _:g2 .
_:g3 rdf:rest rdf:nil .
_:g3 rdf:type rdf:List .
_:g4 rdf:type owl:Class .
_:g4 owl:oneOf _:g5 .
_:g5 rdf:first _:g0 .
_:g5 rdf:rest ~ . # _:j3
_:g5 rdf:type rdf:List .
~ rdf:first _:g1 . # _:j3
~ rdf:rest _:g3 . # _:j3
~ rdf:type rdf:List . # _:j3
```

The two-step process does label the last node.

10.2 Optimizations

This paper has avoided issues to do with optimizing the algorithm presented, but concentrates on the complexity class of the algorithms.

Systems would benefit from merging steps presented here as sequential to reduce the total number of passes of the graph required.

Also, given that most canonicalization is for objects that are small enough to fit in memory, the exposition in terms of sorting files is probably misleading. In-memory structures such as b-trees [1] should be used, and maintaining them in sort order, may be more efficient than permitting them to become unsorted and resorting them as a separate algorithmic step.

The multistep methods in particular are susceptible to optimization. Often the first stage will completely label the blank nodes of the graph. The subsequent stages are then redundant. Moreover, during the first stage we can identify those triples in the graph that require more work. The ones that do not need further work, play no further role in the algorithm, except being included, in a final merge, into the output.

10.3 Multiple Arcs in a Single Comparison

The methods presented, have been constrained for simplicity, to those that can be implemented using Unix `sort` and `awk` over N-Triples.

On larger graphs it may be the case that there are nodes that do not participate in distinctive triples, but for which the set of triples in which they participate is distinctive. A simple example is:

```
_:a <eg:eat> "apple" .
_:a <eg:eat> "pear" .
_:b <eg:eat> "apple" .
_:b <eg:eat> "banana" .
_:c <eg:eat> "pear" .
_:c <eg:eat> "banana" .
```

None of the triples is distinctive, but `_:a` is the only blank node eating both apples and pears.

It may be advantageous to collect all the triples in which a node participates and use the set of labels on those triples as a label to attempt to distinguish the node.

11. PRACTICAL RDF C14N

11.1 Problems concerning rdf:Collection

In example 3 we saw that lists with two or more elements, created with the `rdf:parseType="Collection"` syntax [2], may create a number of blank nodes which:

- Are the subject of a triple with predicate `rdf:type`, and object `rdf:List`.
- Are the subject of a triple with predicate `rdf:first` and object being a blank node.
- Are the subject of a triple with predicate `rdf:rest` and object being a blank node.
- Are the object of a triple with predicate `rdf:rest` and subject being a blank node.

Since none of these triples is distinctive, and the node does not participate in any others, the one pass deterministic labeling will not label such nodes. Moreover the multiple arc label suggested above (section 10.3) does not help.

Given the importance of OWL in the semantic web, and the desire to canonicalize and digitally sign ontologies, this is a disadvantage with one-step methods. Moreover, additional distinctive arcs on such nodes, added as a result of the non-deterministic cheating algorithm, would cause the resulting graph to be one that cannot use the compact `rdf:parseType="Collection"` syntax.

Fortunately the multistep methods will typically distinguish the blank nodes in a typical OWL ontology (possibly requiring the multiple arc label method). This is because while the elements of the list may all be blank, for example a list of restrictions, enough of these elements will participate in distinctive triples, for example, distinctive `owl:hasValue` triples. When adding further distinctive triples it is best to try and exploit the multistep canonicalization and to avoid adding them to the blank nodes carrying the list structure, but instead to add them to the members of the list (e.g. to a blank node of type `owl:Restriction`).

Further study is needed, studying ontologies in say the DAML library [12] and trying a variety of canonicalization techniques over them, to see which work best. If none of these methods proves sufficiently satisfactory, it would be possible to have special treatment of the `rdf:List` construct (for example adding arcs showing the distance between list members, before canonicalization).

11.2 Disadvantages of Graph Modification

The nondeterministic algorithm potentially triples the size of the graph (worst case). This paper is predicated on that potential occurrence not actually happening. The deterministic part of the algorithm will label almost all the nodes in a typical RDF graph

Both of the cheating methods, (deletion or addition) change the RDF graph. In this sense the algorithms presented are not canonicalization algorithms. However, we have argued that for practical applications these changes are acceptable, as long as they happen relatively rarely; they are expected; and that the choice between the deleting method or the adding method has been made appropriately for the application.

A further difficulty is found when canonicalizing multiple subsets of an RDF graph, using the nondeterministic method. The required changes for one such canonicalization will most likely be incompatible with the required changes for another. This can be addressed by using super-properties of `c14n:true` to create the distinctive triples.

12. COMPLEXITY

All the steps in this paper either involve lexicographically sorting a file, for which the best known algorithms are O($n\log n$) [19] or involve stepping through a file line by line and doing either the same simple modification, or a modification involving a table lookup (e.g. step b.ii in section 7). The former steps are O(n), the latter O($n\log n$) (table lookup is O($\log n$), see [1]). Since any particular variant of the techniques involves a constant number of such sort or modification steps the overall complexity is O($n\log n$) where n is the number of triples in the RDF graph.

13. TEST DATA

To see how much nondeterminism is needed to canonicalise typical RDF data we ran the algorithm[4] over the following four sources:

Source	1	2	3	*
RDF test data [16]	130	0	0	0
OWL test data [9]	45	2	0	0
OWL guide [25]	0	0	0	1
DAML ontology library [12]	93	10	5	40

The last line shows that of the files tested from the DAML ontology library 93 were deterministically labeled by the one-step algorithm. A further 10 were deterministically labeled using the two-step algorithm, but that 40 were not fully labeled even with the three step algorithm. The first two lines show that the N-Triples files include with test cases for RDF and OWL are not sufficiently challenging to be good tests for canonicalization.

Thus, at least for ontologies, it appears that direct deterministic methods are not sufficient, and that nondeterministic techniques such as described in section 9.2 are practically necessary.

14. THE FUTURE

The techniques of this paper are not particularly useful in a single stand-alone software product.

They only will be useful as part of a society (a social, legal and technical framework) in which there is widespread use of digital signatures, and wide agreement about what gets signed, and what legal and social obligations such signatures convey.

This agreement will also detail the exact forms of canonicalization used; which will not be precisely what has been articulated in this paper.

Hence, this paper is largely irrelevant unless it feeds into a standardization process.

Further research could be directed towards the fairly practical question of, given RDF data such as we find today, which deterministic classification algorithms are particularly adept; i.e. leaving few nodes unclassified.

Once there is a well-researched answer to this question, then standardizing the techniques of this paper should be straightforward.

15. REFERENCES

[1] G. M. Adel'son-Vel'skii and E. M. Landis. "An algorithm for the Organization of Information". Doklady Akademia Nauk SSSR, vol. 146, 1962, pp. 263-266. English translation in Soviet Mathematics Doklady, vol. 3, pp. 1259-1263.

[4] We used a small prototype implementation [10] written in AWK on a Unix system.

[2] Beckett, D. (ed.). RDF/XML Syntax revised. W3C Working Draft, 2002. http://www.w3.org/TR/2002/WD-rdf-syntax-grammar-20021108/.

[3] Berners-Lee "Architecture", http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html

[4] T. Berners-Lee, R. Fielding, L. Masinter (eds) RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax, Internet Engineering Task Force, 1998.

[5] Berners-Lee Hendler Lassila Semantic Web Scientific American

[6] Paul V. Biron, Ashok Malhotra XML Sxhema Datatypes

[7] John Boyer Canonical XML W3C Recommendation, http://www.w3.org/TR/xml-c14n

[8] Carroll, J.J. "Matching RDF Graphs" in I. Horrocks and J. Hendler (Eds.): ISWC 2002, LNCS 2342, pp. 5-15, 2002.

[9] Carroll, J.J. de Roo, J. Web Ontology Language (OWL) Test Cases, W3C Working Draft, 2002, http://www.w3.org/TR/2002/WD-owl-test-20021024/

[10] Carroll, J.J. Awk scripts for canonicalizing N-Triples, available from (after 22 Nov 2002) http://www-uk.hpl.hp.com/people/jjc/rdf/c14n-impl/

[11] J. Clark and S. DeRose, Editors XML Path Language (XPath) Version 1.0,. World Wide Web Consortium, 16 November 1999. http://www.w3.org/TR/1999/REC-xpath-19991116 .

[12] Mike Dean (librarian), DAML Ontology Library http://www.daml.org/ontologies/

[13] Dean, M. et al. OWL Web Ontology Language 1.0 Reference. W3C Working Draft, 2002. http://www.w3.org/TR/owl-ref/

[14] Dieter Fensel, Christoph Bussler, Alexander Maedche, Semantic Web Enabled Web Services in I. Horrocks and J.Henlder (eds) ISWC 2002

[15] Scott Fortin, The Graph Isomorphism Problem, Technical Report TR 96-20,Department of Computer Science, University of Alberta, 1996. ftp://ftp.cs.ualberta.ca/pub/TechReports/1996/TR96-20/TR96-20.ps.gz

[16] Jan Grant, Dave Beckett, RDF Test Cases, W3C Working Draft, November 2002, http://www.w3.org/TR/2002/WD-rdf-testcases-20021112/

[17] Hayes, P. (ed.). RDF Model Theory. W3C Working Draft, 2002. http://www.w3.org/TR/2002/WD-rdf-mt-20021112/

[18] Klyne, G., Carroll, J. J. Resource Description Framework RDF Concepts and Abstract Syntax. W3C Working Draft, 2002. http://www.w3.org/TR/2002/WD-rdf-concepts-20021108/

[19] D.E.Knuth.The art of computer programming, V.3: Sorting and Searching, 1973.

[20] J. Köbler, U Schöning, J. Torán: The Graph Isomorphism Problem: Its structural complexity, Birkhauser 1993

[21] Clifford Lynch, "Canonicalization: A Fundamental Tool to Facilitate Preservation and Management of Digital Information", D-Lib Magazine September 1999, Volume 5 Number 9, http://www.dlib.org/dlib/september99/09lynch.html

[22] Brendan D. McKay Practical Graph Isomorphism, Congressus Numerantium 30, pp45-87 1981. http://cs.anu.edu.au/~bdm/papers/pgi.pdf

[23] Brendan D. McKay Nauty 1994 http://cs.anu.edu.au/~bdm/nauty/

[24] Ronald C. Read, Derek G.Corneil, Graph Isomorphism Disease, 1977

[25] Michael K. Smith, Deborah McGuinness, Raphael Volz, Chris Welty, Web Ontology Language (OWL) Guide Version 1.0, W3C Working Draft, 2002, http://www.w3.org/TR/2002/WD-owl-guide-20021104/

CLAIMS

1. A method of processing data comprising the steps of:
processing data in accordance with a first set of rules, which operate, *inter alia* to define a stage at which such a processing operation ceases;
applying to the partly-processed data a second set of rules, which operate to modify the data, so that the modified data may be processed in accordance with a third set of rules.

2. A method according to claim 1 wherein the first and third sets of rules are the same.

3. A method according to claim 1 or claim 2 wherein the modification in accordance with the second set of rules modifies the data in a significant manner.

4. A method according to any one of the preceding claims wherein the modification in accordance with the second set of rules modifies the data such that the modified data is processable by the third set of rules.

5. A method according to any one of the preceding claims wherein the data is graphically represented data.

6. A method according to claim 5 any one of the preceding claims wherein the data is an RDF graph.

7. A method according to any one of the preceding claims further comprising the step, performed prior to processing of the data in accordance with the first set of rules, of modifying the data in an insignificant manner.

8. A method according to any one of the preceding claims, wherein the significant modifications include the deletion of significant data.

9. A method according to any one of the preceding claims wherein the significant modifications include the addition of significant data.

10. A method according to claim 9 wherein the significant additions are distinguishable from data which is, prior to performance of any modifications, significant.

11. A method according to any one of the preceding claims wherein the data describes an ontology.

12. A method according to any one of the preceding claims further comprising the step of processing the data in accordance with the third set of rules.

13. A method according to claim 12, further comprising the step, subsequent to the processing of the data in accordance with the third set of rules, of writing or verifying a digital signature establishing authenticity of the data.

14. A method according to any one of the preceding claims, wherein reapplying the method of any one of the preceding claims to data processed in accordance with such a method does not result in any further modification of the data.

1 PATENT NO. : 1234
INVENTOR : DINGLEY
AUTHOR : CHEESEMAN

2 PATENT NO. : 5678
INVENTOR : DINGLEY
AUTHOR : FORMAGGIO

Fig. 1

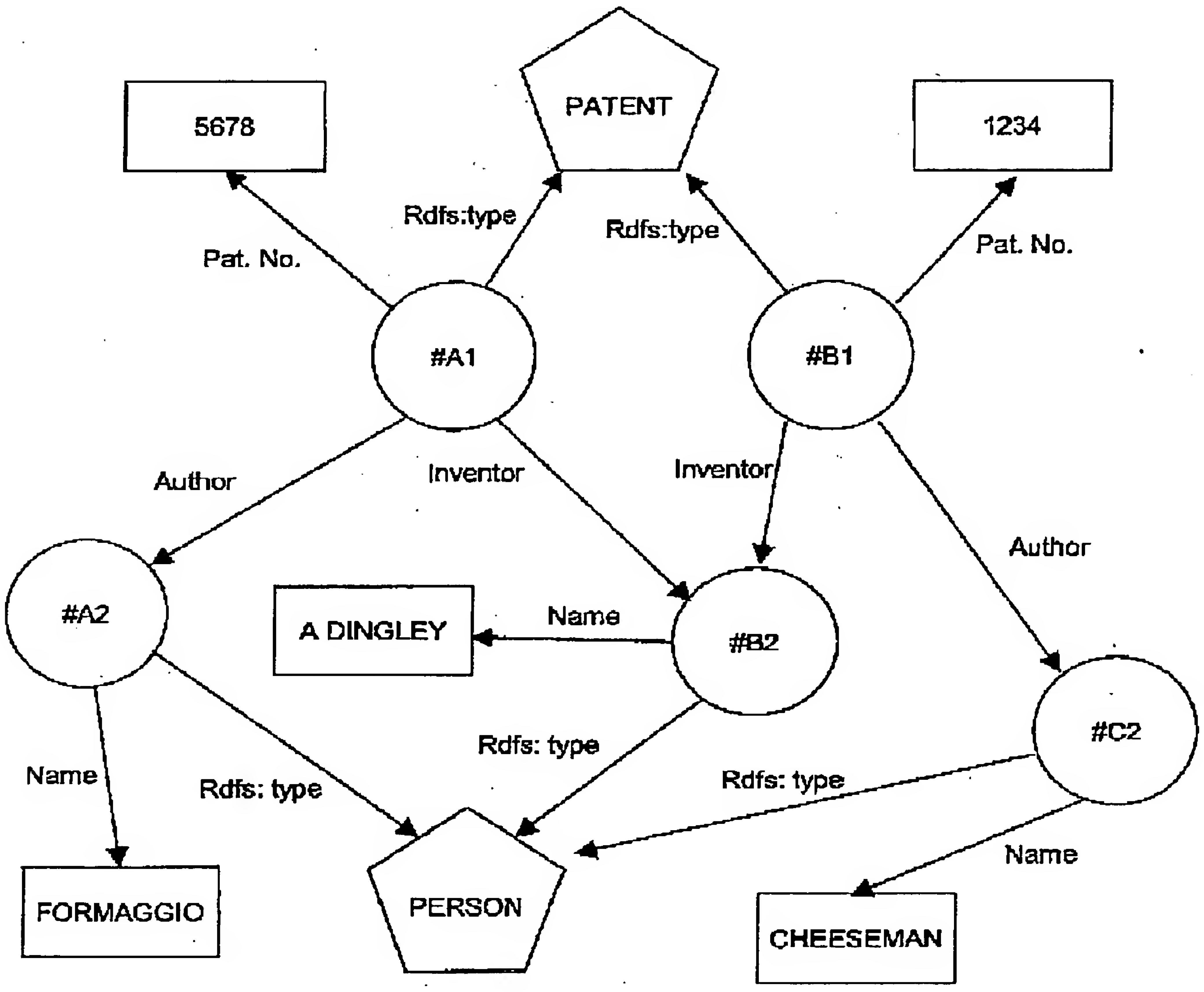

Fig. 2